(19) KOREAN INTELLECTUAL PROPERTY OFFICE

KOREAN PATENT ABSTRACTS

(11)Publication number: 010064226 A
(43)Date of publication of application: 09.07.2001

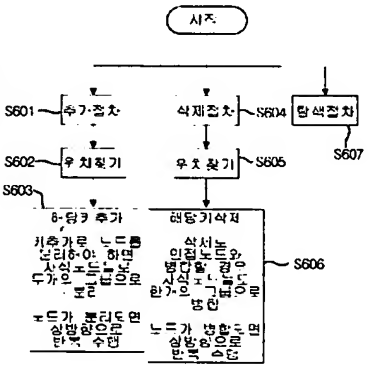
(21)Application number: 990062376
(22)Date of filing: 27.12.1999

(71)Applicant: KOREA ELECTRONICS & TELECOMMUNICATIONS RESEARCH INSTITUTE
(72)Inventor: AHN, SEOK SUN
DO, HAN CHEOL
LEE, HYEONG HO

(51)Int. Cl G06F 17/30

(54) METHOD FOR COMPRESSING, SEARCHING AND INSERTING NEW ITEMS CONSIDERING MEMORY HIERARCHY STRUCTURE

(57) Abstract:



PURPOSE: A method for compressing, searching and inserting new items considering memory hierarchy structure is provided to increase a velocity of a multiple searching tree by using one pointer in one node, thereby identifying (the number of a key x the size of the key + pointer size) with a size of a cache line.

CONSTITUTION: If a new item process is started (S601), a position to be appended by a new item is searched(S602), and the new item is appended to a node thereof. In addition, if the node is full, a medium value is appended to a parent node, and child nodes are divided into and inserted in two consecutive memories(S603). A process to be appended in the parent node is identified with a method to be appended to the current node. If a specific node deletion process is started(S604), a position to be deleted is searched(S605), and the corresponding item is deleted from the corresponding node(S606). As the result of the deletion, if the number of items of the corresponding node is decreased less than the half thereof, the node is integrated to an adjacent node, and the two consecutive memories having children of two nodes are integrated as one consecutive memory. If two adjacent nodes are integrated caused by a deletion, an item indicating the integrated node is deleted. If the number of items of the parent node is decreased, the process is repeated. If a searching process is started, the searching result is obtained by the above (S602) and (S605) stages.

Legal Status
Date of request for an examination (19991227)

Final disposal of an application (registration)
Date of final disposal of an application (20011130)
Patent registration number (1003281290000)
Date of registration (20020227)

(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(51) Int. Cl.
G06F 17/30

(11) 공개번호 특2001-0064226
(43) 공개일자 2001년07월09일

(21) 출원번호 10-1999-0062376
(22) 출원일자 1999년12월27일

(71) 출원인 한국전자통신연구원 오길록
대전 유성구 가정동 161번지

(72) 발명자 안석순
대전광역시유성구송강동한솔아파트202-1109
도한철
대전광역시유성구어은동한빛아파트109-1102
이형호
대전광역시유성구어은동한빛아파트108-1003

(74) 대리인 전영일

심사청구 : 있음

(54) 메모리 계층 구조를 고려한 압축, 탐색 및 새로운 항목삽입 방법

요약

본 발명은 다중 탐색 트리에서 한 노드에서 사용하는 포인터의 개수를 1개만 사용하도록 하여 (키의 개수 × 키의 크기 + 포인터 크기)를 캐시 라인 크기와 일치시켜 다중 탐색 트리를 고속화하고, 키의 크기와 포인터의 크기가 비슷한 네트워크 어드레스 탐색에서 분기수를 2배 가까이 늘려 탐색을 고속화하여 포워딩 엔진의 속도를 높이는 B-tree 계열에 기반한 메모리 계층(Memory Hierarchy) 구조를 고려한 압축, 탐색 및 새로운 항목 삽입 방법을 제공하는데 그 목적이 있다.

본 발명에 따르면, B-tree 계열에 기반한 메모리 계층(Memory Hierarchy) 구조를 고려한 압축 및 탐색 방법에 있어서, 동일한 부모 노드(Parent Node)를 가진 자식 노드(Child Node)들을 연속된 메모리 영역에 배치하여 한 노드당 한 개의 포인터(Pointer)만 사용하여 자식 노드들을 가리키는 것을 특징으로 하는 메모리 계층 구조를 고려한 압축 및 탐색 방법이 제공된다.

대표도

도2b

명세서

도면의 간단한 설명

도 1은 종래의 일반적인 라우터 시스템의 구조도이고,

도 2a부터 도 2c는 본 발명의 일 실시예에 따른 다중 탐색 트리의 대표적인 예인 B-tree를 사용한 예시도로서, 도 2a는 임의의 노드 구조를 보인 예시도이고,

도 2b는 포인터 한 개로 (k+1) way 탐색을 하는 방법을 설명하기 위하여 k = 2인 경우를 나타낸 예시도이고,

도 2c는 도 2a 및 도 2b와 같은 자료 구조를 이용하여 새로운 항목의 추가와 특정 항목의 삭제, 특정 항목의 탐색을 하는 방법을 개념적으로 나타낸 흐름도이고,

도 3은 본 발명의 일 실시예에 따른 메모리 계층 구조를 고려한 압축 및 탐색 방법을 개략적으로 설명하기 위한 흐름도이고,

도 4는 본 발명의 일 실시예에 따른 메모리 계층 구조를 고려한 압축 및 탐색 방법에 있어서, 새로운 자료를 입력하는 과정을 나타낸 흐름도이고,

도 5는 도 4에 도시된 삽입 과정을 상세히 나타낸 흐름도이고,

도 6은 본 발명의 일 실시예에 따른 생성된 트리에 대해서 특정 키를 가진 항목을 찾는 방법을 나타낸 흐름도이다.

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 B-tree 계열에 기반한 메모리 계층(Memory Hierarchy) 구조를 고려한 압축, 탐색 및 새로운 항목 삽입 방법에 관한 것이며, 특히, 다중 탐색 트리에서 한 노드에서 사용하는 포인터의 개수를 1개만 사용하도록 하여 (키의 개수 × 키의 크기 + 포인터 크기)를 캐시 라인 크기와 일치시켜 다중 탐색 트리를 고속화하고, 키의 크기와 포인터의 크기가 비슷한 네트워크 어드레스 탐색에서 분기수를 2배 가까이 늘려 탐색을 고속화하여 포워딩 엔진의 속도를 높이는 B-tree 계열에 기반한 메모리 계층(Memory Hierarchy) 구조를 고려한 압축, 탐색 및 새로운 항목 삽입 방법에 관한 것이다.

데이터 통신망에서는 패킷은 헤더에 자신이 전달될 위치를 표시하는 목적지 주소 필드(Destination Address Field)가 있다. 이 목적지 주소 필드의 값에 따라 다양한 동작을 해 주어야 하기 때문에 이 필드 값에 연관된(Associated) 자료를 빠른 시간에 찾아 내는 것이 중요하다.

이 방법 중의 하나가 다중 탐색 트리(Multiway Search Tree)이고, 대표적인 다중 탐색 트리가 B-tree이다. 다중 탐색 트리는 한 노드(Node)에서 가능한 분기 수(Branching Factor)를 늘려 남겨진 탐색 공간(Search Space)을 크게 줄여 나가는 방법이다.

지금까지 B-tree는 주로 디스크(Disk Storage)에 저장된 데이터를 액세스하는 인덱스(Index)로 사용되었기 때문에 한 노드에서 최대 분기 수(Maximum Branching Factor)를 결정하는 요인은 디스크 블록 크기(Block Size)와 키(Key) 값의 길이에 달려 있어서 키 값을 저장하기 위한 공간을 최소화하여 한 디스크 블록에 최대한 많은 키를 기록하는 것이 과제였다. 데이터 통신망에서 사용되는 패킷의 목적지 주소들은 대개 32 비트 내외의 짧은 것이기 때문에 종래에 고안된 방법으로는 개선의 효과가 없다. 또한, 목적지 주소들은 메인 메모리에 저장되어 있기 때문에 디스크의 블록 크기와 달리 프로세서(Processor)의 캐시 라인 크기(Cache Line Size)에 관한 고려가 중요하다. 프로세서는 메인 메모리에서 한번에 캐시 라인 단위로 읽기 때문에 한 캐시 라인 크기에 최대한 많은 키를 저장하는 방법이 필요하다.

도 1은 종래의 일반적인 라우터 시스템의 구조도로서, 상기 라우터 시스템은 인터페이스(Interface, 110), 포워딩 엔진(Forwarding Engine, 120), 스위칭 패브릭(Switching Fabric, 130) 및 네트워크 프로세서(Network Processor, 140)로 구성된다.

상기 인터페이스(110)는 패킷을 시스템 내부에서 사용하기 위한 형태와 링크(Link)에서 사용될 데이터의 형태 사이를 정합하는 기능을 수행하고, 상기 포워딩 엔진(120)은 수신된 패킷을 분류하여 패킷에 가해질 처리를 결정한다. 예를 들어 패킷의 목적지가 어느 곳인가를 알아내 어느 링크를 통해 내보내야 할지를 결정한다. 상기 스위칭 패브릭(130)은 상기 포워딩 엔진(120)에서 결정된 정보를 이용하여 실제 링크로 데이터를 보내는 기능을 한다.

어드레스 탐색은 어드레스가 사용되는 방법에 따라 Exact Match 및 Longest Prefix Match 방법이 있다. Ethernet MAC 어드레스와 같은 경우는 어드레스의 모든 비트가 일치하여야 하는 Exact Match 방법을 사용해야 하고, IP version 4의 경우에는 찾고자 하는 키와 첫 번째 비트부터 가장 길게 일치하는 자료를 찾는 방법을 사용하여야 한다. Longest Prefix Matching이 Exact Matching에 비하여 어렵다는 것이 일반적인 사실이다. 따라서, 지금부터는 Longest Prefix Matching을 위주로 설명한다.

지금까지 Longest Prefix Matching 문제를 해결하기 위하여 가장 많이 사용되었던 방법이 Patricia Tree이다.

Patricia Tree에서 특정한 키를 찾는 방법은 루트 노드(Root Node)에서 시작하여 키의 각 비트의 값 0,1에 따라 트리를 잎 노드(Leaf Node)를 만날 때까지 따라 간다. 이 과정에서 만난 노드 중 가장 나중에 만난 표시가 되어있는 노드가 찾고자 하는 키와 일치되는 정보를 가지고 있는 노드가 된다. 이 방법의 문제점은 최악의 경우 키의 길이와 같은 횟수 만큼의 노드를 억세스해야 한다는 것이다.

Patricia Tree의 경로 길이(Path Length)를 줄이기 위한 방법이 몇 가지 제안되었는데, 대표적인 방법이 Level Compress 방법이다.

Patricia Tree는 경로 길이를 줄이기 위하여 분기가 일어나지 않는 경로를 축약하는 방법을 사용하는 반면에 Level Compress 방법에서는 분기가 많은 노드들이 밀집한 부분에서 노드를 축약한다.

또한, Perlman은 Longest Prefix Matching 문제를 Exact Matching 문제로 변환하는 방법을 제시하였다. 이와 같은 문제 변환을 통해 기존에 Exact Matching에 사용되던 잘 알려진 다양한 방법들을 Longest prefix matching 문제에 사용할 수 있다. Lampson은 이와 같은 방법에 기존에 잘 알려진 다중 탐색 트리(Multiway Search Tree)를 사용하여 Longest Prefix Matching이 필요한 인터넷 프로토콜(IP)에 적용하였다.

Lampson이 다중 탐색 트리를 사용한 이유는 현대적인(Modern) 프로세서 구조들이 일반적으로 취하는 계층적 메모리 구조(Memory Hierarchy)를 최대한 활용하는 것이다. 심화되는 메모리 속도와 프로세서 속도차를 극복하기 위하여 점점 대용량의 캐시가 프로세서에 탑재되고 있다. 이 캐시의 동작은 임의의 메모리 어드레스가 엑세스되면 이 메모리 어드레스를 포함하는 캐시라인(Cache Line) 전체가 한꺼번에 프로세서의 캐시로 복사되고 캐시 내에 있는 데이터는 프로세서의 속도로 처리된다. 따라서, 다중 탐색 트리의 한 노드를 한 캐시 라인 크기로 만들면 추가적인 메인 메모리 엑세스 없이 분기 수를 대폭 늘일 수 있게 된다.

종래의 다중 탐색 트리는 디스크와 메인 메모리 사이에서 메모리 계층 구조를 고려하여 고안된 방법이기 때문에 저속의 메모리와 상대적으로 고속인 메인 메모리를 고려하여 한 번에 디스크에서 메인 메모리로 읽어 들인 디스크 블록을 이용하여 추가적인 저속의 디스크 엑세스 없이 분기 수를 메인 메모리의 속도로 늘리는 방법이다. 이러한 다중 탐색 트리의 대표적인 예가 B-tree와 그 변형들이다.

B-tree가 사용된 일반적인 경우는 디스크에 저장된 데이터베이스(Database)를 억세스하기 위하여 역시 디스크에 저장된 인덱스파일(Index File)을 구성하는 데 사용된다. 데이터베이스를 억세스하기 위한 키는 문자열(Character String)로 구성되어 이 노드에서 비교 결과에 따라 연결될 다음 노드를 가리키기 위한 포인터의 길이에 비해 아주 많은 메모리를 차지하게 된다. 따라서, 한 번에 억세스 가능한 디스크 블록 안에 최대한 많은 노드를 넣기 위하여 키가 차지하는 메모리의 양을 줄이기 위한 방법들이 고안되었다.

발명이 이루고자하는 기술적 과제

본 발명은 상기와 같은 종래 기술의 문제점을 해결하기 위하여 안출된 것으로서, 다중 탐색 트리에서 한 노드에서 사용하는 포인터의 개수를 1개만 사용하도록 하여 (키의 개수 × 키의 크기 + 포인터 크기)를 캐시 라인 크기와 일치시켜 다중 탐색 트리를 고속화하고, 키의 크기와 포인터의 크기가 비슷한 네트워크 어드레스 탐색에서 분기수를 2배 가까이 늘려 탐색을 고속화하여 포워딩 엔진의 속도를 높이는 B-tree 계열에 기반한 메모리 계층(Memory Hierarchy) 구조를 고려한 압축, 탐색 및 새로운 항목 삽입 방법을 제공하는데 그 목적이 있다.

발명의 구성 및 작용

앞서 설명한 바와 같은 목적을 달성하기 위한 본 발명에 따르면, B-tree 계열에 기반한 메모리 계층(Memory Hierarchy) 구조를 고려한 압축 및 탐색 방법에 있어서, 동일한 부모 노드(Parent Node)를 가진 자식 노드(Child Node)들을 연속된 메모리 영역에 배치하여 한 노드당 한 개의 포인터(Pointer)만 사용하여 자식 노드들을 가리키는 것을 특징으로 하는 메모리 계층 구조를 고려한 압축 및 탐색 방법이 제공된다.

또한, 삽입하고자 하는 새로운 항목을 추가할 공간이 없으면, 동일한 부모 노드들을 가진 노드들을 연속된 메모리 영역에 두기 위하여 미리 영역을 확보해 놓은 후, 기설정된 위치 이후 노드들을 뒤로 이동시켜 놓는 제 1 단계와; 분리되어 새로 생성될 노드 위치를 설정하고, 새로운 키가 분할된 노드 중 왼쪽 노드, 중간 노드 및 오른쪽 노드에 위치한 노드에 들어갈 것인지를 판단하는 제 2 단계와; 상기 제 2 단계에서의 판단 결과, 중간 노드에 해당하면, 추가될 항목을 바로 부모 노드에 추가하고, 왼쪽 노드에 해당하면, 새로운 항목을 왼쪽 노드에 추가하며, 오른쪽 노드에 해당하면, 왼쪽 노드에서 옮겨올 항목들을 옮긴 후, 추가할 노드를 기설정한 위치에 넣는 제 3 단계와; 왼쪽 노드와 오른쪽 노드에 있는 항목 수를 수정한 후, 새로운 연속된 메모리 영역을 할당받아 오른쪽 노드에 할당된 자식 노드들을 새로 할당한 메모리로 옮기는 제 4 단계를 포함하여 이루어진 것을 특징으로 하는 B-tree 계열에 기반한 메모리 계층(Memory Hierarchy) 구조를 고려한 새로운 항목 삽입 방법이 제공된다.

또한, 컴퓨터에, 동일한 부모 노드(Parent Node)를 가진 자식 노드(Child Node)들을 연속된 메모리 영역에 배치하여 한 노드당 한 개의 포인터(Pointer)만 사용하여 자식 노드들을 가리키는 제 1 단계와; 새로운 노드가 추가되어 특정 노드가 분할되어야 하는 경우, 상기 자식 노드들도 같이 연속된 두 개의 메모리 영역으로 분할하고, 한 노드에 한 개의 캐시 라인 크기를 적용함으로써, 분기수를 늘리는 제 2 단계와; 특정 항목이 삭제되어 특정 노드가 통합되어야 하는 경우, 상기 자식 노드들도 같이 연속된 두 개의 메모리 영역으로 통합하고, 한 노드에 한 개의 캐시 라인 크기를 적용함으로써, 분기수를 늘리는 제 3 단계를 포함하여 이루어진 것을 실행시킬 수 있는 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록 매체가 제공된다.

또한, 컴퓨터에, 삽입하고자 하는 새로운 항목을 추가할 공간이 있으면, 새로운 항목을 삽입하는 제 1 단계와; 삽입하고자 하는 새로운 항목을 추가할 공간이 없으면, 동일한 부모 노드들을 가진 노드들을 연속된 메모리 영역에 두기 위하여 미리 영역을 확보해 놓은 후, 기설정된 위치 이후 노드들을 뒤로 이동시켜 놓는 제 2 단계와; 분리되어 새로 생성될 노드 위치를 설정하고, 새로운 키가 분할된 노드 중 왼쪽 노드, 중간 노드 및 오른쪽 노드에 들어갈 것인지를 판단하는 제 3 단계와; 상기 제 3 단계에서의 판단 결과, 중간 노드에 해당하면, 추가될 항목을 바로 부모 노드에 추가하고, 왼쪽 노드에 해당하면, 새로운 항목을 왼쪽 노드에 추가하며, 오른쪽 노드에 해당하면, 왼쪽 노드에서 옮겨올 항목들을 옮긴 후, 추가할 노드를 기설정한 위치에 넣는 제 4 단계와; 왼쪽 노드와 오른쪽 노드에 있는 항목 수를 수정한 후, 새로운 연속된 메모리 영역을 할당받아 오른쪽 노드에 할당된 자식 노드들을 새로 할당한 메모리로 옮기는 제 5 단계를 포함하여 이루어진 것을 실행시킬 수 있는 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록 매체가 제공된다.

아래에서, 본 발명에 따른 양호한 일 실시예를 첨부한 도면을 참조로 하여 상세히 설명하겠다.

도 2a부터 도 2c는 본 발명의 일 실시예에 따른 다중 탐색 트리의 대표적인 예인 B-tree를 사용한 예시도로서, 도 2a는 임의의 노드 구조를 보인 예시도이다.

도 2a에서와 같이 한 노드는 (k+1) way B-tree의 k 개의 키와 노드 내의 몇 개의 키가 있는지를 나타내는 $n(1 \le m \le k)$, 가장 왼쪽 자식(Child) 노드를 가리키는 포인터 p로 구성된다. 따라서, 32 바이트 캐시 라인을 가진 프로세서에서는 키의 크기가 32 비트이고, 포인터의 크기가 16인 경우 8 way B-tree를 구성할 수 있다.

도 2c는 포인터 한 개로 (k+1) way 탐색을 하는 방법을 설명하기 위하여 k = 2인 경우를 나타낸 예시도이다. B-tree에서는 잎 노드에서 시작하여 루트 노드 방향으로 새로운 노드가 생성되어 간다. 이때 새로운 노드에 의하여 가리켜지는 자식 노드들을 연속된 메모리에 할당하도록 한다. 키 15와 키 16으로 이루어진 노드 14의 자식 노드들은 연속된 메모리 영역 18에 할당한다. 노드 14의 부모 노드의 모든 자식 노드들도 마찬가지로 연속된 메모리 영역 19에 할당한다. 키 값 15와 키 값 16 사이의 키 값을 가진 노드들은 포인터 17이 가리키는 연속된 메모리에서 두 번째 노드 20에 기록된다.

도 2c는 도 2a 및 도 2b와 같은 자료 구조를 이용하여 새로운 항목의 추가와 특정 항목의 삭제, 특정 항목의 탐색을 하는 방법을 개념적으로 나타낸 흐름도로서, 이를 상세히 설명하면 다음과 같다.

먼저, 스텝 S601에서, 새로운 항목 추가 절차가 시작되면, 스텝 S602에서, 새로운 항목이 추가될 위치를 찾고, 스텝 S603에서, 그 노드에 새로운 항목을 추가한 후, 그 노드가 이미 Full 이면, 중간 값을 부모 노드에 추가하는 동작을 하고, 자식 노드들을 두 개의 연속된 메모리에 나누어 담는다. 부모 노드에 추가되는 과정도 현재 노드에 추가되는 방법과 동일한 방법이 적용된다.

스텝 S604에서, 특정 노드 삭제 절차가 시작되면, 스텝 S605에서, 삭제하여야 할 노드의 위치를 찾은 후, 스텝 S606에서, 해당 항목을 해당 노드에서 삭제한다. 삭제의 결과로 해당 노드의 항목 수가 반 이하로 줄어들면, 인접한 노드와 통합하고, 그 두 노드의 자식들이 들어 있는 두 개의 연속한 메모리를 한 개의 연속한 메모리로 통합한다. 삭제한 항목으로 인하여 두 인접 노드를 통합할 경우에는 그 부모 노드에서 통합된 노드를 가리키던 항목을 삭제한다. 삭제로 인하여 부모 노드의 항목 수가 반 이하로 줄어들면 같은 방법을 반복한다. 이와 같은 방법으로 B-tree에서 한 개의 포인터만을 사용하여 다중 탐색 트리를 구현할 수 있다.

스텝 S607에서, 탐색 절차가 시작되면, 위에서 서술한 스텝 S602 및 스텝 S605의 절차에 의하여 탐색 결과를 얻을 수 있다.

도 3은 본 발명의 일 실시예에 따른 메모리 계층 구조를 고려한 압축 및 탐색 방법을 개략적으로 설명하기 위한 흐름도로서, 이를 상세히 설명하면 다음과 같다.

먼저, 스텝 S701에서, 루트 노드를 포함하는 연속된 메모리의 위치를 가리키는 변수 root를 nil로 초기화한 후, 스텝 S702에서, 연속된 메모리 안의 루트 노드의 위치를 가리키는 변수 rootloc을 0으로 초기화한다.

이어서, 스텝 S703에서, 새로운 키 x가 입력되면, 스텝 S704에서, 상기 x가 0인지 여부를 판단한다. 이때 x = 0는 입력을 끝내는 조건이다.

상기 스텝 S704에서의 판단 결과, x가 0이 아니면, 스텝 S705에서, 확인을 위하여 x를 출력하고, 스텝 S706에서, 새로운 키를 트리에 추가하는 search를 수행한다. 이때 매개 변수 x는 추가할 키, root와 rootloc은 현재 루트 노드의 위치, h는 키 x의 추가로 새로운 루트 노드를 생성하여야 하는지 여부를 나타낸다. ue는 새로 생성될 루트 노드에서의 키 값, up 및 uloc은 이 키 값이 가리켜야 하는 자식 노드의 위치를 나타낸다.

이어서, 스텝 S707에서, 새로 추가되어야 할 노드가 있다고 표시되면 스텝 S708부터 스텝 S711까지 새로운 루트 노드를 만든다.

한편, 상기 스텝 S704에서의 판단 결과, x가 0이면, 즉, 입력을 끝내는 조건을 만족시키지 아니하면, 스텝 S112부터 스텝 S119까지의 과정을 통하여 입력된 데이터에 의하여 생성된 다중 탐색 트리를 시험 데이터를 입력 받아 탐색을 수행한다. 이때 스텝 S115의 find는 주어진 x를 찾는 루틴이다. 매개 변수 x는 찾고자 하는 키, root와 rootloc은 루트 노드의 위치, ue는 해당 키에 대한 정보를 돌려주는 변수이다. 찾고자 하는 키가 있을 경우 복귀 값이 true이고, 이때 관련 정보는 ue에 저장된다.

도 4는 본 발명의 일 실시예에 따른 메모리 계층 구조를 고려한 압축 및 탐색 방법에 있어서, 새로운 자료를 입력하는 과정을 나타낸 흐름도로서, 이를 상세히 설명하면 다음과 같다.

먼저, 스텝 S801에서, 현재 조사되고 있는 노드 포인터가 잎 노드에서 가리키는, 즉 nil인지 여부를 판단한다.

상기 스텝 S801에서의 판단 결과, nil이면, 스텝 S802부터 스텝 S804 과정을 통하여 새로운 키가 입력된다는 것을 표시하고, 입력할 키 값과 딸린 정보를 초기화한다. 여기서는 설명을 위하여 딸린 정보로 해당 키가 몇번이나 억세스되는 지를 저장하고 있다.

또한, 상기 스텝 S801에서의 판단 결과, nil이 아니면, 스텝 S805부터 스텝 S807 과정을 통하여 한 노드에서 입력된 키 x의 위치를 찾기 위한 이진 탐색을 수행한 후, 스텝 S808에서, 키가 현재 노드에 포함되어 있는지 여부를 판단한다.

상기 스텝 S808에서의 판단 결과, 키가 현재 노드에 포함되어 있으면, 스텝 S809 및 스텝 S810에서, 이미 있는 키 값에 대해서는 억세스 횟수를 표시하는 count 정보를 증가시킨다.

상기 스텝 S808에서의 판단 결과, 키가 현재 노드에 포함되어 있지 아니하면, 스텝 S811 및 스텝 S812에서, 입력한 키가 있을 수 있는 구간을 가진 자식 노드 포인터를 가지고 search를 재귀적으로 수행한 후, 스텝 S813에서, 함수 호출 결과, 복귀값 h로 새로운 노드가 현재 노드에 추가되어야 하는지 여부를 판단한다.

상기 스텝 S813에서의 판단 결과, 추가되어야 하면, 스텝 S814에서, 삽입 과정을 수행한 후, 복귀하고, 추가되어야 할 필요가 없으면, 바로 복귀한다.

도 5는 도 4에 도시된 삽입 과정을 상세히 나타낸 흐름도로서, 이를 상세히 설명하면 다음과 같다.

먼저, 스텝 S901에서, 현재 노드에 새로운 항목을 추가할 공간이 있는지 여부를 판단한다.

상기 스텝 S901에서의 판단 결과, 새로운 항목을 추가할 공간이 있으면, 스텝 S902부터 S905 과정을 통하여, 해당 위치에 항목을 추가한다.

상기 스텝 S901에서의 판단 결과, 새로운 항목을 추가할 공간이 없으면, 스텝 S906에서, 동일한 부모 노드들을 가진 노드들을 연속된 메모리 영역에 두기 위하여 미리 영역을 확보해 놓은 후, 적당한 위치 이후 노드들을 뒤로 이동시켜 놓는다. 이어서, 스텝 S907에서, 분리되어 새로 생성될 노드 위치 b를 설정하고, 스텝 S908 및 스텝 S909에서, 새로운 키가 분할된 노드 중 왼쪽 노드, 중간 노드 및 오른쪽 노드에 위치한 노드에 들어갈 것인지를 판단한다.

상기 스텝 S908 및 스텝 S909에서의 판단 결과, 중간 노드에 해당하면, 추가될 항목의 위치가 한 가운데 이어서 현재 노드에 추가할 필요 없이 바로 부모 노드에 추가되는 경우이므로, 스텝 S910부터 스텝 S912 과정을 통하여 매개 변수만 저장한다.

상기 스텝 S908 및 스텝 S909에서의 판단 결과, 왼쪽 노드에 해당하면, 스텝 S913부터 스텝 S917 과정을 통하여 새로운 항목을 왼쪽 노드에 추가한다.

상기 스텝 S908 및 스텝 S909에서의 판단 결과, 오른쪽 노드에 해당하면, 스텝 S919부터 스텝 S925 과정을 통하여 비슷한 방법으로 왼쪽 노드에서 옮겨올 항목들을 옮기고, 추가할 노드를 적정 위치에 넣는다.

이어서, 스텝 S926 및 스텝 S927에서, 왼쪽 노드와 오른쪽 노드에 있는 항목 수를 수정한 후, 스텝 S928부터 스텝 S931 과정은 상기 스텝 S906에서부터 시작하여 수행할 절차가 새로운 노드를 생성하기 위하여 기존 노드 하나를 둘로 분할하였으므로, 원래 노드가 가리키던 자식 노드의 연속된 메모리들도 두개의 연속하는 메모리에 저장된 자식 노드들로 분할해야 하기 때문에 새로운 연속된 메모리 영역을 할당받아 오른쪽 노드에 할당된 자식 노드들을 새로 할당한 메모리로 옮긴다.

이와 같은 방법으로 잎 노드로부터 루트 노드로 새로운 항목들이 전달되어 가면서 트리의 경로가 길어지는 경우가 생긴다.

도 6은 본 발명의 일 실시예에 따른 생성된 트리에 대해서 특정 키를 가진 항목을 찾는 방법을 나타낸 흐름도로서, 이를 상세히 설명하면 다음과 같다.

먼저, 스텝 S1001에서, 호출한 노드가 잎 노드인지를 판단하여, 잎 노드이면, 스텝 S1002에서, 키 없음을 출력하고 종료하며, 잎 노드가 아니면, 스텝 S1003부터 스텝 S1005의 과정을 통하여, 루트 노드로부터 시작하여 각 노드를 따라가면서 키가 위치하는 포인터를 찾는다.

이어서, 스텝 S1006에서, 일치하는 항목이 있으면, 스텝 S1007 및 스텝 S1008에서, 찾은 키를 저장하고, 표시한 후, 종료하고, 일치하는 항목이 없으면, 스텝 S1009 및 스텝 S1010에서, 자식 노드에서 찾는다.

새로운 항목을 추가하는 방법과 생성된 트리에서 주어진 키를 찾는 방법만을 설명하였는데 주어진 항목을 삭제하는 방법도 새로운 항목을 추가하는 방법과 같은 개념을 사용한다. 즉, 임의의 항목이 삭제되어 인접 노드와 통합이 필요하면, 그 두 노드의 자식 노드들도 두 개의 연속한 메모리에 있던 것을 한 개의 연속한 메모리에 통합하면 된다.

상기와 같은 본 발명은 컴퓨터로 읽을 수 있는 기록 매체로 기록되고, 컴퓨터에 의해 처리될 수 있다.

발명의 효과

앞서 상세히 설명한 바와 같이 본 발명은 다중 탐색 트리에서 한 노드에서 사용하는 포인터의 개수를 (키 개수 + 1) 개가 필요하던 종래의 기술에서, 1개만 사용하도록 하여 (키의 개수 × 키의 크기 + 포인터 크기)를 캐시 라인 크기와 일치시켜 다중 탐색 트리를 고속화하여 키의 크기와 포인터의 크기가 비슷한 네트워크 어드레스 탐색에서 분기수를 2배 가까이 늘려 탐색을 고속화하여 포워딩 엔진의 속도를 높이는 효과가 있다.

또한, 부수적으로 각각의 노드에서 (키의 최대 개수 + 1 개)가 필요하던 포인터를 한 개만 사용하게 되므로 필요한 메모리도 대폭 절감된다. 메모리 요구량이 작아진 부수 효과에 따라 2차 캐시에 들어가는 비율이 높아지게 되어 추가 속도 향상 효과를 얻을 수 있다.

한편, 메모리 대역폭(Memory Bandwidth)은 크지만 셋업 시간이 오래 걸리는 블록 모드(Block Transfer) 또는 버스트 모드(Burst Mode)를 가지는 메모리와 대형의 캐시 라인 사이즈를 가지는 프로세서나 ASIC 사이에서 효과가 크다. 또, 내장메모리(Embedded Memory)를 가지는 ASIC의 경우나 내장 메모리를 가지지 못하고 대량의 외부 메모리가 필요하며 SRAM을 사용하지 못하고 버스트 모드를 지원하는 대용량 메모리를 사용해야 할 경우에도 적합하다.

이상에서 본 발명에 대한 기술 사상을 첨부 도면과 함께 서술하였지만 이는 본 발명의 가장 양호한 일 실시예를 예시적으로 설명한 것이지 본 발명을 한정하는 것은 아니다. 또한, 이 기술 분야의 통상의 지식을 가진 자이면 누구나 본 발명의 기술 사상의 범주를 이탈하지 않는 범위 내에서 다양한 변형 및 모방이 가능함은 명백한 사실이다.

(57) 청구의 범위

청구항 1. B-tree 계열에 기반한 메모리 계층(Memory Hierarchy) 구조를 고려한 압축 및 탐색 방법에 있어서,

동일한 부모 노드(Parent Node)를 가진 자식 노드(Child Node)들을 연속된 메모리 영역에 배치하여 한 노드당 한 개의 포인터(Pointer)만 사용하여 자식 노드들을 가리키는 것을 특징으로 하는 메모리 계층 구조를 고려한 압축 및 탐색 방법

청구항 2. 제 1 항에 있어서,

새로운 노드가 추가되어 특정 노드가 분할되어야 하는 경우, 상기 자식 노드들도 같이 연속된 두 개의 메모리 영역으로 분할하고, 한 노드에 한 개의 캐시 라인 크기를 적용함으로써, 분기수를 늘리는 것을 특징으로 하는 메모리 계층 구조를 고려한 압축 및 탐색 방법.

청구항 3. 제 1 항에 있어서,

특정 항목이 삭제되어 특정 노드가 통합되어야 하는 경우, 상기 자식 노드들도 같이 연속된 두 개의 메모리 영역으로 통합하고, 한 노드에 한 개의 캐시 라인 크기를 적용함으로써, 분기수를 늘리는 것을 특징으로 하는 메모리 계층 구조를 고려한 압축 및 탐색 방법.

청구항 4. 삽입하고자 하는 새로운 항목을 추가할 공간이 없으면, 동일한 부모 노드들을 가진 노드들을 연속된 메모리 영역에 두기 위하여 미리 영역을 확보해 놓은 후, 기설정된 위치 이후 노드들을 뒤로 이동시켜 놓는 제 1 단계와;

분리되어 새로 생성될 노드 위치를 설정하고, 새로운 키가 분할된 노드 중 왼쪽 노드, 중간 노드 및 오른쪽 노드에 위치한 노드에 들어갈 것인지를 판단하는 제 2 단계와;

상기 제 2 단계에서의 판단 결과, 중간 노드에 해당하면, 추가될 항목을 바로 부모 노드에 추가하고, 왼쪽 노드에 해당하면, 새로운 항목을 왼쪽 노드에 추가하며, 오른쪽 노드에 해당하면, 왼쪽 노드에서 옮겨올 항목들을 옮긴 후, 추가할 노드를 기설정한 위치에 넣는 제 3 단계와;

왼쪽 노드와 오른쪽 노드에 있는 항목 수를 수정한 후, 새로운 연속된 메모리 영역을 할당받아 오른쪽 노드에 할당된 자식 노드들을 새로 할당한 메모리로 옮기는 제 4 단계를 포함하여 이루어진 것을 특징으로 하는 B-tree 계열에 기반한 메모리 계층(Memory Hierarchy) 구조를 고려한 새로운 항목 삽입 방법.

청구항 5. 제 4 항에 있어서,

삽입하고자 하는 새로운 항목을 추가할 공간이 있으면, 현재 노드에 공간이 없으므로 현재 노드를 두 개의 노드로 나누어 담고, 새로운 노드를 위한 공간을 확보하여 새로운 항목을 삽입하는 제 5 단계를 더 포함하여 이루어진 것?? 특징으로 하는 B-tree 계열에 기반한 메모리 계층(Memory Hierarchy) 구조를 고려한 새로운 항목 삽입 방법.

청구항 6. 컴퓨터에,

동일한 부모 노드(Parent Node)를 가진 자식 노드(Child Node)들을 연속된 메모리 영역에 배치하여 한 노드당 한 개의 포인터(Pointer)만 사용하여 자식 노드들을 가리키는 제 1 단계와;

새로운 노드가 추가되어 특정 노드가 분할되어야 하는 경우, 상기 자식 노드들도 같이 연속된 두 개의 메모리 영역으로 분할하고, 한 노드에 한 개의 캐시 라인 크기를 적용함으로써, 분기수를 늘리는 제 2 단계와;

특정 항목이 삭제되어 특정 노드가 통합되어야 하는 경우, 상기 자식 노드들도 같이 연속된 두 개의 메모리 영역으로 통합하고, 한 노드에 한 개의 캐시 라인 크기를 적용함으로써, 분기수를 늘리는 제 3 단계를 포함하여 이루어진 것을 실행시킬 수 있는 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록 매체.

청구항 7. 컴퓨터에,

삽입하고자 하는 새로운 항목을 추가할 공간이 있으면, 새로운 항목을 삽입하는 제 1 단계와;

삽입하고자 하는 새로운 항목을 추가할 공간이 없으면, 동일한 부모 노드들을 가진 노드들을 연속된 메모리 영역에 두기 위하여 미리 영역을 확보해 놓은 후, 기설정된 위치 이후 노드들을 뒤로 이동시켜 놓는 제 2 단계와;

분리되어 새로 생성될 노드 위치를 설정하고, 새로운 키가 분할된 노드 중 왼쪽 노드, 중간 노드 및 오른쪽 노드에 위치한 노드에 들어갈 것인지를 판단하는 제 3 단계와;

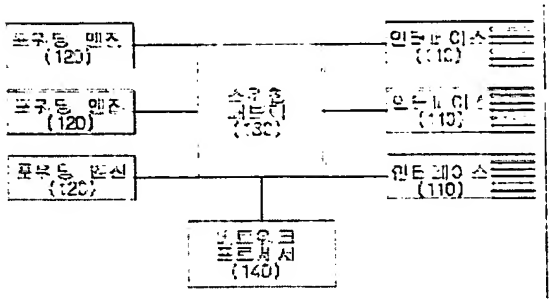
상기 제 3 단계에서의 판단 결과, 중간 노드에 해당하면, 추가될 항목을 바로 부모 노드에 추가하고, 왼쪽 노드에 해당하면, 새로운 항목을 왼쪽 노드에 추가하며, 오른쪽 노드에 해당하면, 왼쪽 노드에서 옮겨

을 항목들을 옮긴 후, 추가할 노드를 기설정한 위치에 넣는 제 4 단계와;

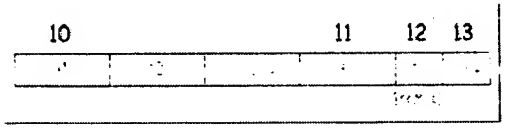
왼쪽 노드와 오른쪽 노드에 있는 항목 수를 수정한 후, 새로운 연속된 메모리 영역을 할당받아 오른쪽 노드에 할당된 자식 노드들을 새로 할당한 메모리로 옮기는 제 5 단계를 포함하여 이루어진 것을 실행시킬 수 있는 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록 매체.

도면

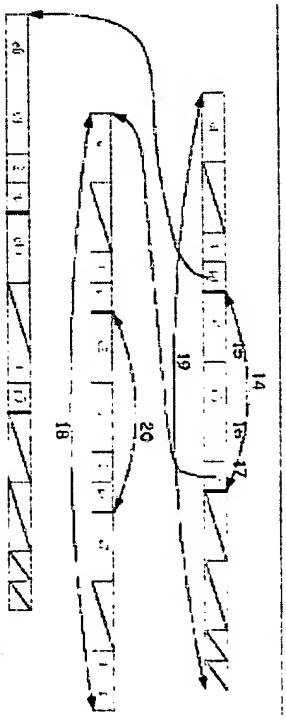
도면1



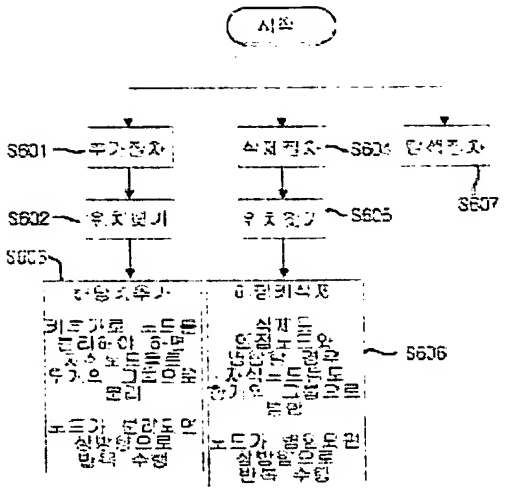
도면2a



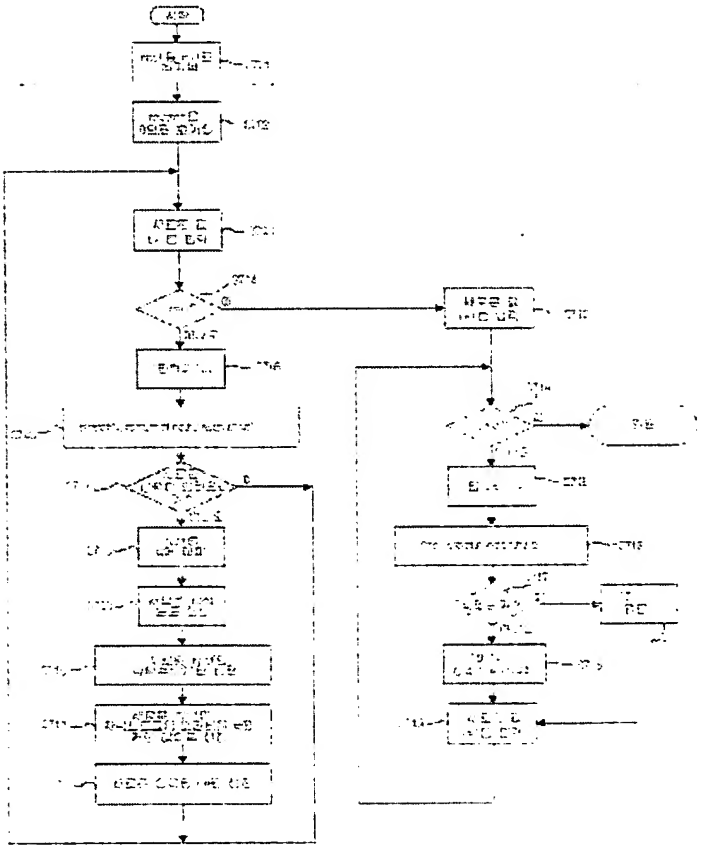
도면2b



도면5

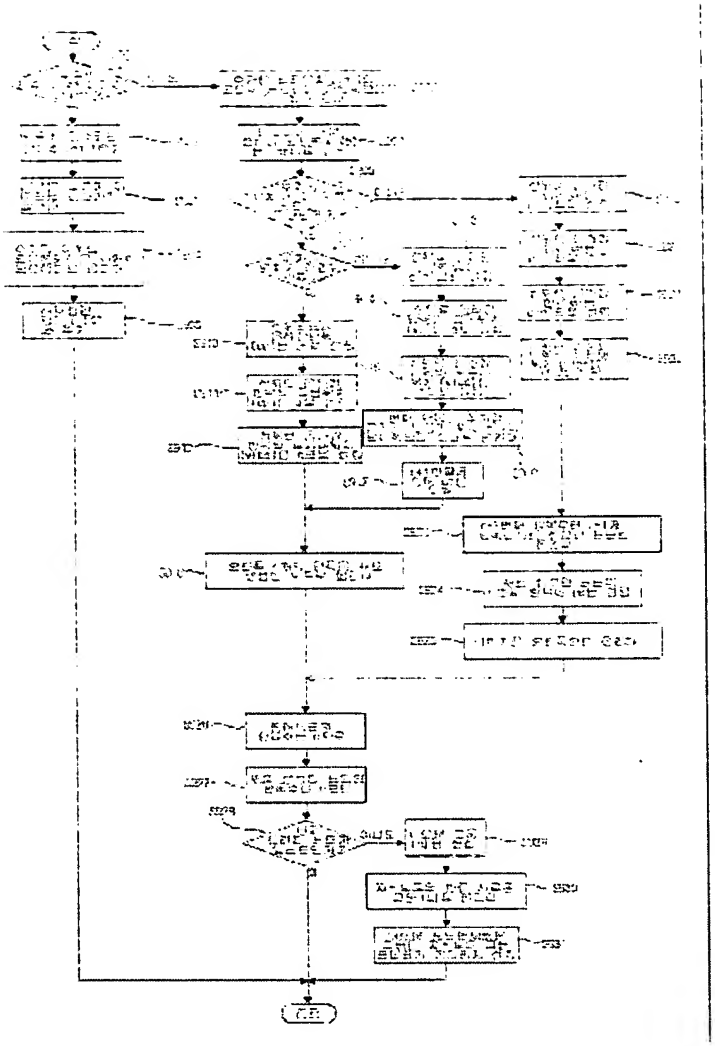


도면6



도면3

도면5



도면